CS575 Parallel Computing Project#6
OpenCL Array Multiplication and Reduction

Shangjia Dong

[1]School of Civil and Construction Engineering, Oregon State University

dongs@oregonstate.edu

1. Array Multiply and Array Multiply-Add

1.1.

Question: Tell what machine you ran this on.

Response: I ran this on rabbit.

- 2 E5-2630 Xeon Processors
- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- CPU(s): 32
- Thread(s) per core: 2; Core(s) per socket: 8
- Socket(s): 2
- 64 GB of memory; 2 TB of disk

1.2.

Question: Show the tables and graphs

Table 1. OpenCL Array Multiplication Performance

LOCAL_SIZE	GLOBAL_SIZE	#WORK_GROUPS	MegaMultsPerSecond
8	1K	128	0.013
16	1K	64	0.016
32	1K	32	0.013
64	1K	16	0.017
128	1K	8	0.017
256	1K	4	0.017
512	1K	2	0.017
8	32K	4096	0.376
16	32K	2048	0.585
32	32K	1024	0.609
64	32K	512	0.437
128	32K	256	0.432
256	32K	128	0.629

Table 2. OpenCL Array Multiplication Performance (continue)

LOCAL_SIZE	GLOBAL_SIZE	#WORK_GROUPS	MegaMultsPerSecond
512	32K	64	0.433
8	512K	65536	1.228
16	512K	32768	1.617
32	512K	16384	2.139
64	512K	8192	1.977
128	512K	4096	1.956
256	512K	2048	2.357
512	512K	1024	2.094
8	1024K	131072	1.662
16	1024K	65536	2.84
32	1024K	32768	3.68
64	1024K	16384	3.37
128	1024K	8192	2.953
256	1024K	4096	3.412
512	1024K	2048	3.851
8	4096K	524288	2.53
16	4096K	262144	4.32
32	4096K	131072	6.635
64	4096K	65536	8.315
128	4096K	32768	8.73
256	4096K	16384	8.472
512	4096K	8192	8.724
8	8192K	1048576	2.703
16	8192K	524288	5.024
32	8192K	262144	7.752
64	8192K	131072	10.177
128	8192K	65536	11.231
256	8192K	32768	12.108
512	8192K	16384	11.307
8	10240K	1310720	2.809
16	10240K	655360	5.031
32	10240K	327680	8.446
64	10240K	163840	11.512
128	10240K	81920	12.757
256	10240K	40960	12.042
512	10240K	20480	12.74

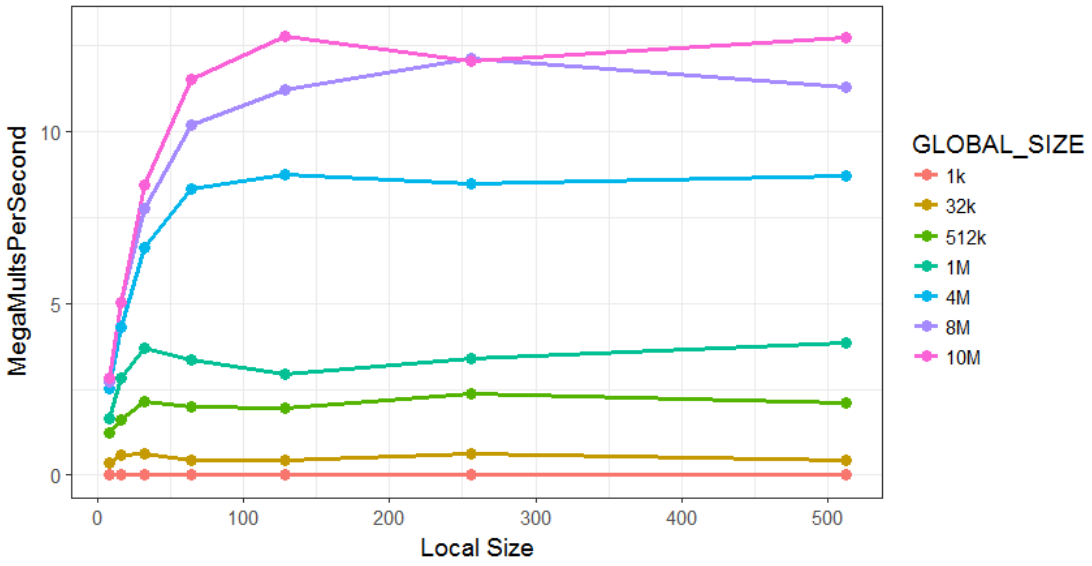

Figure 1. OpenCL Array Multiplication Performance

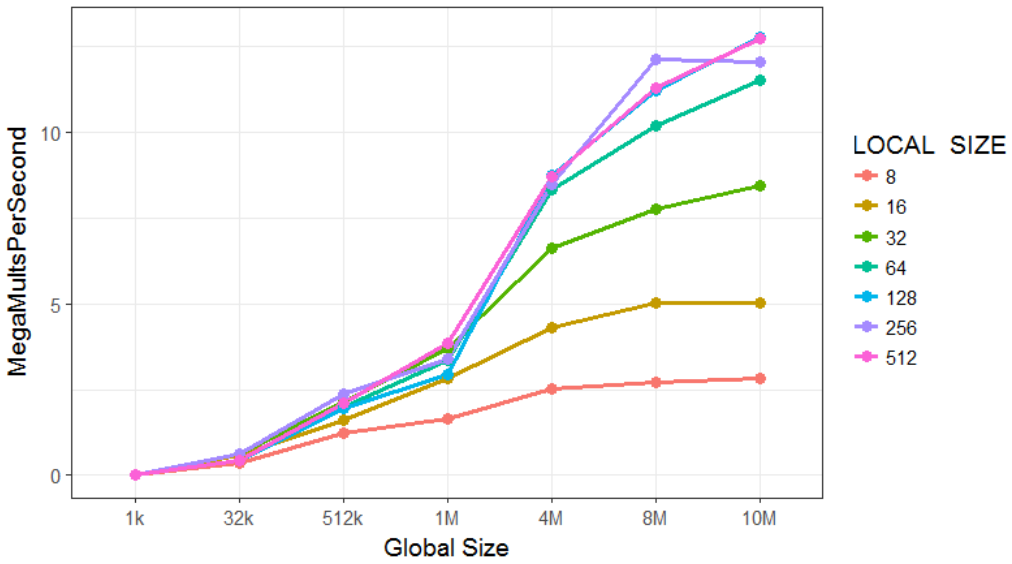

Figure 2. OpenCL Array Multiplication Performance

Table 3. OpenCL Array Multiplication-Add Performance

GLOBAL_SIZE	LOCAL_SIZE	NUM_WORK_GROUPS	MegaMultsAddPerSecond
1K	8	128	0.013
1K	16	64	0.018
1K	32	32	0.018
1K	64	16	0.02
1K	128	8	0.021
1K	256	4	0.014
1K	512	2	0.015
32K	8	4096	0.378
32K	16	2048	0.596
32K	32	1024	0.42
32K	64	512	0.482
32K	128	256	0.412
32K	256	128	0.37
32K	512	64	0.564
512K	8	65536	1.311
512K	16	32768	1.767
512K	32	16384	2.142
512K	64	8192	2.315
512K	128	4096	1.825
512K	256	2048	2.322
512K	512	1024	2.296
1024K	8	131072	1.842
1024K	16	65536	2.716
1024K	32	32768	2.705
1024K	64	16384	3.18
1024K	128	8192	3.763
1024K	256	4096	3.007
1024K	512	2048	3.103
4096K	8	524288	2.325
4096K	16	262144	4.038
4096K	32	131072	5.641
4096K	64	65536	6.631
4096K	128	32768	6.866
4096K	256	16384	6.835
4096K	512	8192	6.187
8192K	8	1048576	2.665
8192K	16	524288	4.66
8192K	32	262144	7.576
8192K	64	131072	9.604
8192K	128	65536	9.306
8192K	256	32768	10.219
8192K	512	16384	9.427

Table 4. OpenCL Array Multiplication-Add Performance

GLOBAL_SIZE	LOCAL_SIZE	NUM_WORK_GROUPS	MegaMultsAddPerSecond
10240K	8	1310720	2.679
10240K	16	655360	4.949
10240K	32	327680	7.818
10240K	64	163840	9.141
10240K	128	81920	10.481
10240K	256	40960	10.661
10240K	512	20480	10.634

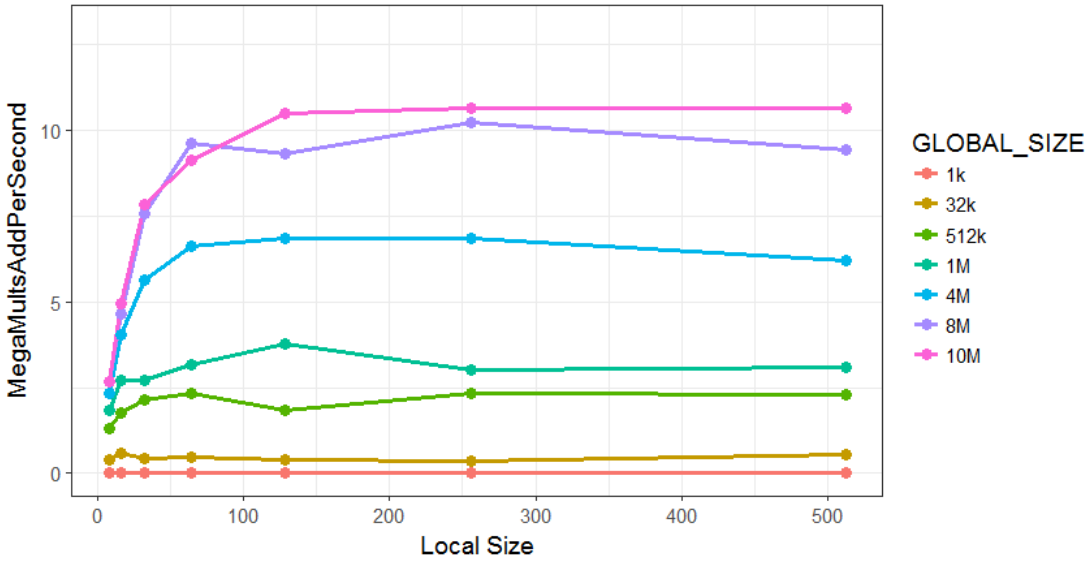

Figure 3. OpenCL Array Multiplication-Add Performance

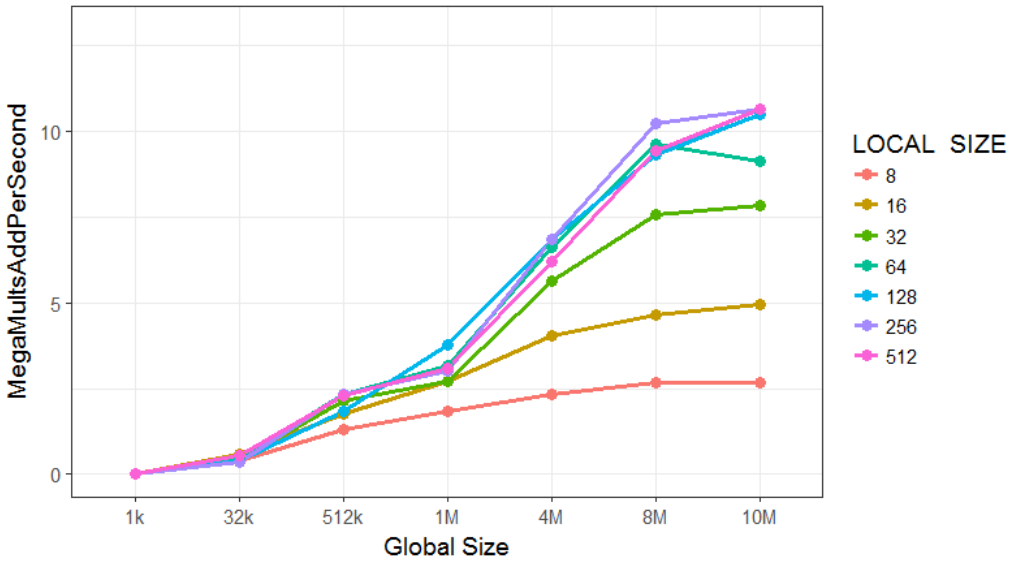

Figure 4. OpenCL Array Multiplication-Add Performance

1.3.

Question: What patterns are you seeing in the performance curves?

Response: Array Multiply: First, as in Figure 1, the performance increases as the local size (work group size) increase, but after a "sweet spot", the performance stops increasing and tends to be steady. In this case, after work group size of 128, the performance stops increasing.

Second, as in Figure 2, when the global size (array size) increases, the performance increases, the increase slows down when passing a certain array size. Also, when the array size and group size are both large, the performance are getting similar, I suspect they will converge together eventually. Surprisingly, the local size (work group size of 512) is not the best all the time.

Array Multiply + Add: Looking at the Figure 3 4, they presents the similar overall pattern as we described in Array multiply case. It further confirms that not necessarily the larger the local size (work groups size) the better, because group size of 128 seems to has a better performance compares to other local sizes. I suspect this is because the work group size exceeds the number of the processing element on the computing unit.

1.4.

Question: Why do you think the patterns look this way?

Response: When there the local size (work group size) is two small, e.g., 8, there are more processing elements in each compute unit than 8. Thus if there are 32 processing elements, we am only using 8 at a time, a lot of computing time is wasted. As the local size (work group size) increases, the performance is getting better, which means we are fully using the resources. It seems after local size = 128, there are no much changes. I think it indicates 128 is a good group size. That is to say, when the work group size is too small, it is not worth it because you don't get enough work done to overcome the overhead to set it up.

1.5.

Question: What is the performance difference between doing a Multiply and doing a Multiply-Add?

Response: The figures are plotted at the same scale. As you can see, in the same case, the performance of Multiply+Add is lower than the performance of Multiply. The patterns are the similar, but the performance dropped 1-2 at each step. In addition, when we are doing Multiply only, local size (work group) of 512 has similar performance as 128, however, in Multiply+Add case, local size (work group) of 512 has better performance than 128.

1.6.

Question: What does that mean for the proper use of GPU parallel computing?

Response: Since the performance Multiply+Add is lower than Multiply only case, one potential useful suggestion for further GPU parallel computing would be, if there are more calculations are added, either the local size (work group size) or the global size (array size) should be increased to achieve the desired performance, but this is effective only in a certain range.

2. Multiply+Reduce Application

2.1.

Question: Show this table and graph

Response: In my case, I only have local size of 4, 8, 16, 32 working, so here I only recorded the performance of these four. But I hope it can still tells the pattern.

Table 5. Multiply + Reduce Performance

GLOBAL_SIZE	LOCAL_SIZE	NUM_WORK_GROUPS	MegaMultsPerSecond
1K	4	256	0.007
1K	8	128	0.007
1K	16	64	0.007
1K	32	32	0.007
32K	4	8192	0.197
32K	8	4096	0.194
32K	16	2048	0.214
32K	32	1024	0.137
512K	4	131072	0.748
512K	8	65536	1.114
512K	16	32768	1.635
512K	32	16384	1.655
1024K	4	262144	0.826
1024K	8	131072	1.317
1024K	16	65536	2.026
1024K	32	32768	2.978
4096K	4	1048576	0.892
4096K	8	524288	1.49
4096K	16	262144	2.449
4096K	32	131072	4.063
8192K	4	2097152	0.909
8192K	8	1048576	1.517
8192K	16	524288	2.624
8192K	32	262144	4.473
10240K	4	2621440	0.913
10240K	8	1310720	1.551
10240K	16	655360	2.649
10240K	32	327680	4.539

2.2.

Question: What pattern are you seeing in this performance curve?

Response: As you can see in the Figure 5, in general, when the local size (work group size) increase, the performance increases, except for the cases when global size (array size) is very small, such as 1k and 32k. When global size = 512k, the performance stops increasing at group size of 16. Although this is not enough data point in this figure,

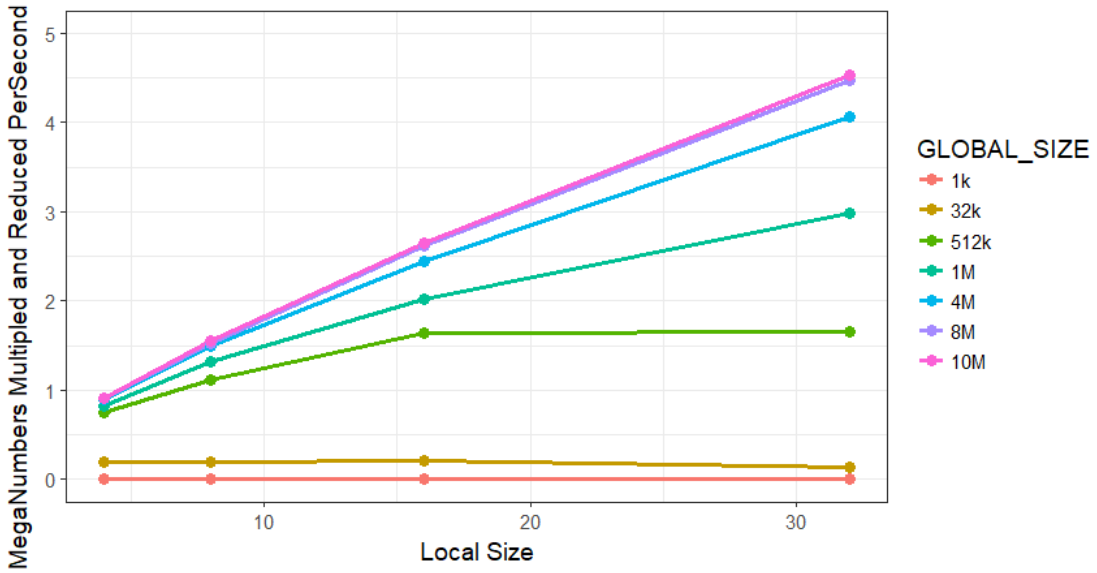

Figure 5. OpenCL Array Multiplication+Reduce Performance

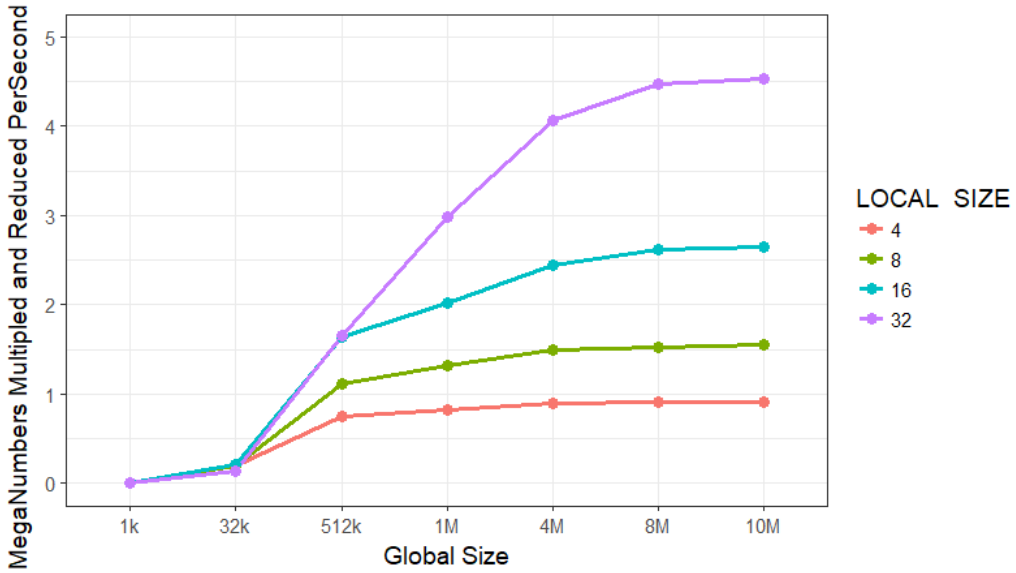

Figure 6. OpenCL Array Multiplication+Reduce Performance

I suspect as the global size (array size) increase, this sweet point also becomes larger. Also, as the global size becomes larger, the performance also get closer, I suspect there is a point that performance stops increasing as the gloabl size increases.

On the other side, the argument we made can be validated in Figure 6. For example, you can clearly see that when the global size (array size) larger than 32k, the difference in performance starts to show up. As the global size (array size) increases, the performance increases and stay steady after a point. This point varies depends on the value of the local size (work group size) as well. As the local size (work group size) increase, this point shifts to further right (becomes larger).

2.3.

Question: Why do you think the pattern looks this way?

Response: When the global size is too small, i.e. 1k, 32k etc., the performance is poor. Because there are not many benefits in doing reduce because the overhead of setting up is larger than the benefit we can get from the reduce. As the global size (array size) increases, the multiplication can really use that reduce and make the simulation faster. In addition, the local size should also be in a reasonable size. When the local size (work group size) is very small, there are many processing elements on the computing unit are wasted, thus the performance is very poor.

2.4.

Question: What does that mean for the proper use of GPU parallel computing?

Response: Before doing the multiply and reduce, first check if the problem size is large enough, then decide if it is worthy to do the reduce. In addition, in setting up the reduce, using a reasonable work group size will boost the performance.